DISTRIBUTED VIDEO PROCESSING SYSTEM USING SPARK

Siddhant Garg [*1] **Sridhama Prakhya** [*1]

ABSTRACT

In this work, we tackle the problem of building a scalable video processing system, where the goal is to extract, store, and retrieve characteristic visual features from videos. We will use a machine learning model, called CLIP, for supporting large set of objects and action classes and to find a list of high-precision tags for every video. To effectively handle large video datasets, the proposed system leverages PySpark on top of PyTorch for enabling the automatic distribution of workloads across different processes and for making fast inferences on GPU. A MongoDB database server hosts the extracted set of tags for each video to support fast indexing and retrieval.

1 INTRODUCTION

Nowadays, a lot of video content is being constantly generated across different platforms which have necessitated the invention of efficient processing, indexing, and retrieval video processing systems. For extracting useful information from the videos, initial approaches used readily available information like meta-data, textual information, or keyword annotation for indexing videos. Later on, people started diving deeper and extracting visual information using heuristic-based frame-based descriptors (Cha et al., 2017) to capture motion activity (Liu et al., 2017), texture (Zhou et al., 2020), shape, or color (Minaei et al., 2017) which describes the low-level visual content using pixel values of the image frames. Furthermore, with the advent of machine learning models, it became possible to make more high-level and diverse sets of predictions on video frames. However, these methods are very slow considering the large number of frames involved, even in a small video. On top of this, the machine learning models support limited batch size in one forward pass which requires efficient batching, sampling and aggregating the extracted tags across all frames in a single video. Therefore, it is essential to develop scalable and distributed video processing systems with machine learning support for high quality video information retrieval.

Motivation: Parallelizing video processing for machine learning applications is very difficult and manually intensive work with standalone frameworks like PyTorch. Although, the PyTorch library is highly optimized to support for making parallel machine learning inferences but the APIs for efficient frame sampling is still lacking in many ways. The major issues in the existing implementations is that they use randomized algorithms to sample a fixed amount of frames for every video to ensure a fixed batch size. This approach is only suitable for training machine learning models on video datasets because repeated iterations will ensure that the model gets trained on most of the frames. Furthermore, the video-level atomicity is lost while training because most of the times all the videos are concatenated together and the dataloader iterates on ”one single video” repeatedly. But in our case, we want to run only a single forward pass to extract useful information from videos without leaving out any useful frames while sampling as well as preserve the atomicity of each video while storing that information in a distributed database. In this regard, the existing systems with machine learning support are not capable for designing large-scale video processing, indexing and retrieval engines.

Problem Statement: In this work, our primary aim is to design an efficient data mining pipeline for a large collection of raw, and unlabeled videos. We have used PySpark on top of PyTorch to distribute the computations across different workers in a fault tolerant and consistent manner. The PySpark library is fairly easy to use as it hides the distribution process in the backend and it can be integrated with a number of different frameworks while supporting pythonic development. We have also used PySpark to integrate MongoDB database server in the same pipeline so that the extracted tags can be directly pushed to the database. The proposed system can parallelize the mining process while maintaining the atomicity of each input and it can leverage the large amount of GPU clusters that are nowadays, readily available in many institutions to perform parallel inferences.

Each worker process performs frame sampling to extract relevant video frames, dense feature extraction using batch-inference, feature tagging with scores for all the supported classes for each sampled frame and aggregation of tags

[*]Equal contribution [1]Department of Computer Science, University of Massachusetts Amherst. Correspondence to: Siddhant Garg <siddhantgarg@umass.edu>, Sridhama Prakhya <sprakhya@cs.umass.edu>.

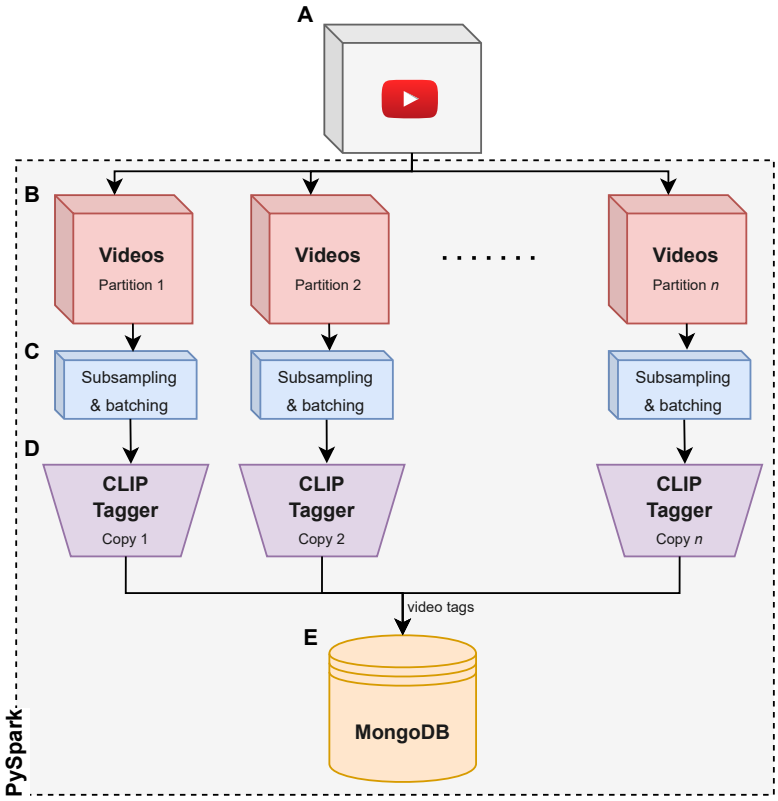

Figure 1. A high-level overview of our Spark-based parallel video tagger. **A.** Corpus of YouTube videos. **B.** Partitions of input videos ($1...n$). **C.** Subsampling videos at 1 FPS followed by batching frames. **D.** Tagging batched video frames using CLIP. Note that all instances of the CLIP tagger are identical for different partitions. **E.** Video tags are pushed to a global remote MongoDB database instance for further analytics.

across the video for getting high-precision classes. We have used Decord[1] for reading videos and the frame-sampling step. Decord is one of the most efficient video readers that support in-memory storage of frames. While most video readers like PyAv[2], OpenCV[3], FFmpeg[4], and Pytorch's VideoReader[5] seek each frame sequentially, Decord can extract any frame by only using the frame index thus making the sampling process faster without the need for implementing loops and skipping consecutive frames.

Generally machine learning models are required to be trained on a specific set of classes in a supervised manner for making highly accurate predictions. In order to support the tagging of video datasets from randomly diverse sources (social media, YouTube, or streaming platforms), either multiple machine learning models need to be trained on different datasets with similar distribution or a single multi-domain model needs to be trained on a large dataset to handle the distribution shift in the inputs. Moreover, we would need annotated data for training the models that would further increase the cost of building such a system.

However, recently a new class of machine learning models (CLIP (Radford et al., 2021), DINO (Caron et al., 2021), LSeg (Li et al., 2022)) have come up that do not require explicit training on desired problems and are able to support robust zero-shot predictions using relevant signals. One of the models is CLIP that uses natural language supervision to learn visual features. It consists of a text transformer encoder and an image transformer encoder that joinly learns their embedding space in order keep the related textual and visual embeddings closer with respect to the distance metric while keeping all other negative pairs farther apart. Furthermore, it was trained on diverse set of classes and encodes a large set of concepts in its parameters. In order to make zero-shot predictions we only require a desired class and hand-engineered textual prompts without training the model again for the new class. Therefore it is a suitable model that can be integrated with our pipeline to tag videos across from domains for a large number of classes. In this work, we have used classes from the ImageNet (Deng et al.,

[1] https://github.com/dmlc/decord
[2] https://github.com/PyAV-Org/PyAV
[3] https://docs.opencv.org/3.4/d8/dfe/classcv_1_1VideoCapture.html
[4] https://ffmpeg.org/
[5] https://pytorch.org/vision/stable/generated/torchvision.io.VideoReader.html

2009) and Kinectics-700 (Kay et al., 2017) datasets and available hand engineered prompts from the CLIP's GitHub repository [6]. The extracted tags for each frame's features are aggregated and the top scoring tags are sampled for each of the datasets. In this work we took 5 highest scoring tags from ImageNet classes and 5 highest scoring tags from the Kinetics classes out of all the predicted scores in a single video for their respective datasets. Note that there can a same class can occur more than once in the set of top 5 scoring classes from different frames so the final collection of tags might be less than 10. Even though, we decreased the diversity of predictions, this ensures high-precision in our predictions as whatever gets predicted is correct with high probability.

Finally, the extracted tags list is stored in the MongoDB database to enable retrieval by querying. The database is a NoSQL database with horizontal scalability. It is most suitable for our usecase because of efficient indexing of the list of tags for each video without introducing sparsity in the table and due to the backend optimizations of the database, the query processing and retrieval is also very fast.

2 Related Work

Our work can be categorized under the umbrella of content-based video retrieval (CBVR) systems. The aim of traditional CBVR systems is to support representing, modeling, indexing, retrieving, browsing or searching information stored in a multimedia databases (Saoudi & Jai-Andaloussi, 2021). The scope of our work touches upon most of the above described properties. Recently, there have been extensive works on CBVR in order to support most flexible video-content retrieval. However, the existing systems lack in many aspects. Either, they still use heuristic based features of processing videos, or they use complicated frame sampling procedures involving Grouping of Frames (Saoudi & Jai-Andaloussi, 2021), fuzzy c-means clustering (Aote & Potnurwar, 2019), clustering based on color channels (Das et al., 2018), using moving object detection and Peak Signal-to-Noise Ratio (Luo et al., 2018) or using spatiotemporal slices (Zhang et al., 2016). All these methods require additional multi-step modules in the pipeline that increases the computation time in frame sampling with no guarantee of preserving the useful frames. In contrast, we sampled frames at 1 Frames per second (FPS) rate for downsampling while preserving useful frames.

Perhaps the work of (Saoudi & Jai-Andaloussi, 2021) is closely related to our work in developing a distributed CBVR system. They use Apache Storm[7] as a distributed real-time computation system which is based on the master-slave architecture where the master node assigns tasks to each of the workers. However, they use computationally extensive frame sampling using Group of pictures to first group the closest frames and select only 1 key frame from the group. This leads to almost 8 different processing steps at worker nodes. Furthermore, they have only performed their experiments on very short video clips on Hollywood2 dataset (Wang & Schmid, 2013) which are very short video clips (less than 1 minute) with 24 FPS rate whereas we use longer videos with atleast 4-5 mins with 30 FPS rates or higher.

Other CBVR system is VideoQ (Chang et al., 1998) that support spatio-temporal queries for retrieving video clips like basketball players or skiers. However, they implemented a java-based interactive query interface for specifying multi-object queries whereas we are using a distributed database server for the the same. Moreover, the VideoQ system's query server contains multiple databases, each for indexing a single specific feature like color, motion, shape, or texture. It is clear that they use low-level features and complex queries to support video retrievals while we use machine learning models to explicitly store high-level object and action classes and support simple queries. Other related work (Ilyas & Rehman, 2019) use a machine learning model for extracting useful classes from videos but there system requires explicit training of the models on the supported classes and it is not scalable while our system can be easily scaled out across different machines.

Furthermore, most popular machine learning frameworks like PyTorch also support video reading and frame sampling but they are designed in order to make the training processing more faster and simpler instead of the inference process. The VideoReader API[8] requires iterating through every frame in the video whereas, the VideoFramesDataset[9] used randomized algorithms to sample a fixed number of frames for every video. This is not suitable because every video is of different duration and either we would need to change the number of sampled frames for every video to ensure robust frame sampling or if the number of frames is fixed, then there will be loss of relevant information as the interval time between the sampled frames will be large in longer videos.

Our work addresses all of the above mentioned shortcomings in the existing works and at the same time propose a novel distributed system for state-of-the-art video data mining. We will now descibe each of the components of our system in detail along with the results in the following sections.

[6] https://github.com/openai/CLIP

[7] https://github.com/apache/storm

[8] https://pytorch.org/vision/main/auto_examples/plot_video_api.html

[9] https://video-dataset-loading-pytorch.readthedocs.io/en/latest/

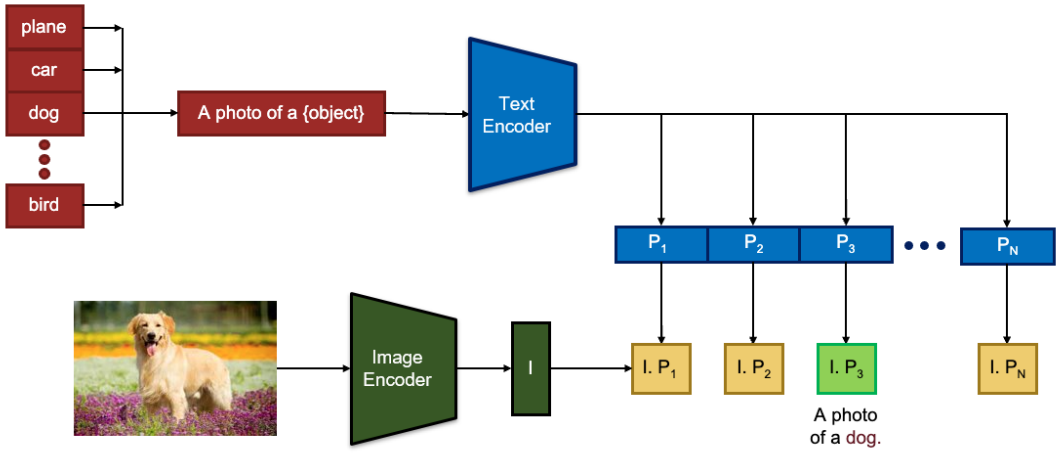

Figure 2. A high-level overview of CLIP's zero-shot tagging pipeline. Given an input image, the image encoder computes the dense visual features. Similarly, textual prompt features are computed using the text encoder. The predicted class corresponds to the the maximum dot product scores between the image embedding and the prompt embedding.

3 APPROACH

The overview of our approach is described in Figure 1, where we have a large corpus of videos (A), and it is partitioned across difference workers using PySpark (B). Each video is then processed sequentially afterwards where features are extracted for the sampled framed (C) using the CLIP model and those features are tagged in a zero-shot manner (D). The output for each video is a list of highest scoring tags. The lists from all the partitions is concatenated and then pushed to the MongoDB server (E).

For a single GPU system with enough memory, different workers initiate different copies of the CLIP model on the GPU. Initially, when the number of workers increases, the inference time reduces but as the number of workers keeps increasing the process becomes slow because more workers compete for limited available resources. For multi-GPU settings, we can easily extend our approach to host partitions on different GPUs, with each GPU hosting exactly the number of workers that maximize the throughput on a single GPU.

Video reader and Frame sampling: We used Decord to read each video for faster in-memory processing of frames. Decord is efficient, flexible and it provides convenient video slicing methods based on a wrapper on top of hardware accelarated video decoders like FFMPEG/LibAV and Nvidia Codecs. It does not require decoding videos to frames, supports frames access using $get\_batch()$ and frame index list and it 2x faster than OpenCV VideoCapture and Pyav Container.

To support frame sampling and frames batching for machine learning model inference, we used PyTorch's `Dataset` class on top of Decord. Each `Dataset` instance method takes a video path as input and extract the video's FPS, duration and the total number of frames in each video using Decord. Using these three quantities, the indices of the frames after every 1 second interval can be computed and those frames were then read and stored in-memory in the class variables. The class's `getitem(i)` method will return the frame at the i^{th} index after applying a transformation function and making it suitable for the CLIP model's input. The combination of Decord and PyTorch's Dataset class leveraged the highly optimized APIs from individual components to maximize our model's input throughput. We believe that our subsampling frequency of 1 FPS captures sufficient temporal context without missing important visual cues. Furthermore, our subsampling process helps reduce the input data size, thereby improving inference speed.

3.1 Zero-shot video tagging with CLIP

Figure 2 describes the high-level overview of how zero-shot predictions can be made using a single textual prompt, a set of classes and a query image.

Both the image and text transformer encoders of the CLIP model generate embeddings for a given image and a textual prompt respectively such that if the text describes the image, then the embeddings will be closer in embedding space, otherwise they will be distant. Therefore, if we have an image of a dog, its visual embedding will be closer to the prompt embedding "`This is a photo of a dog`" than the prompt embedding of "`This is a photo of a cat`". We can generalize this to more classes by replacing the `label` with the class name in the prompt— "`This is a photo of a {label}`"— and take the class with highest cosine similarity as the zero-shot prediction.

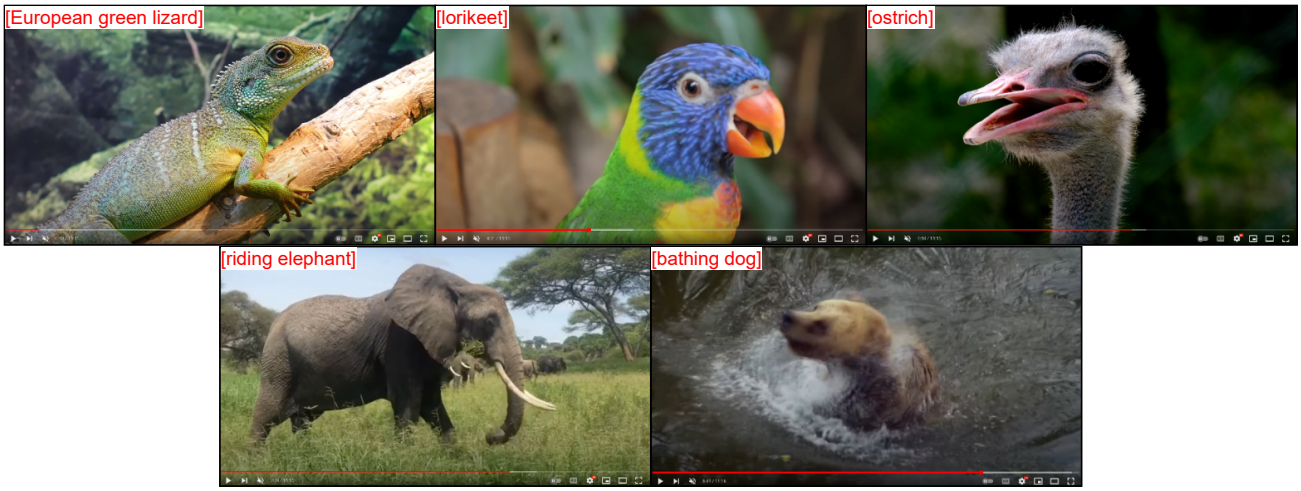

Figure 3. Example tags and their corresponding frames inferred by our system using the CLIP model. Note that tag prediction capability is not perfect due to object ambiguities which may be hard for even humans to distinguish correctly. Note that the last frame is incorrectly tagged as `[bathing dog]` while it is actually a bear. This is because of general limitations of the machine learning model's incapability of handling noisy or blurry inputs itself, rather than shortcomings of our pipeline. Video source: `https://www.youtube.com/watch?v=muczNvx9fgg` (video not from MERLOT Reserve dataset)

Since the transformer models can be highly sensitive to different prompts, there are many hand-engineered prompts that are proposed in the CLIP paper for datasets like ImageNet and Kinectics-700. For calculating the similarity score between the image and class tag, we append each class name to all the text prompts and take the average of the cosine similarity as the final class score for more robust zero-shot prediction. This is repeated for all the classes. The highest-scoring class can be taken as the final prediction.

For our video tagging use case, we used the classes and prompts for two different datasets: ImageNet (80 prompts and 1,000 classes) and Kinetics-700 (28 prompts and 700 action classes). For each video, we computed the class scores for every frame, extracted at 1 FPS. To ensure high-precision tagging, we took the top-5 highest scoring classes across all the frames for each dataset (ImageNet/Kinetics-700).

3.2 Storing video tags and querying them using PySpark SQL

After tagging, we stored the list of video tags in a remote MongoDB database instance using PySpark SQL for basic data insertion and querying operations. We chose MongoDB primarily due to its ability to scale horizontally and also its support for semi-structured data, such as variable-length video tag arrays in our case.

4 Experiments and Results

4.1 Data

We used a small portion of the MERLOT Reserve dataset, which is a large-scale unlabelled dataset of YouTube videos where each video comes with a high-level category. In total, we sampled 500 videos due to disk and time constraints. We present inference time on these 500 videos for different numbers of partitions on a single GPU.

4.2 Downloading and storing data

We obtained a subset of videos from MERLOT Reserve (Zellers et al., 2022). Given the size and modality (video) of our dataset, we can either do this in an online or offline fashion. By downloading the video on demand, rather than storing the dataset of videos on disk, we can reduce the spatial footprint of our system. However, this comes with the additional overhead of internet latency, which can, in turn, factor into inference latency. Ideally, if disk space is not a constraint, it is preferred to download all videos in the dataset and store them for offline access, which is what we ended up doing.

4.3 Evaluation

We evaluated the proposed system based on how well it scales to the size of the dataset and the time it takes to extract the relevant tags with varying numbers of parallel processes.

Figure 4. Representative frames from videos retrieved upon running a conditional query. The conditional query, executed through PySpark SQL, was designed to retrieve videos that contain both "`driving car`" AND "`car mirror`". Note that frame one in Video A and frame two in Video B both contain "`driving car`", while the second and third frames from Video A and Video B respectively both contain "`car mirror`"—although it's slightly out of frame, but nevertheless visible in Video B. Two other top-ranked tags generated by the system were: "`news anchoring`" and "`changing gear in car`".
Both videos depicted in this figure are from the MERLOT Reserve dataset. Video A source: `https://www.youtube.com/watch?v=QamPzj0Xzek` / Video B source: `https://www.youtube.com/watch?v=1kvdhlA_BXY`

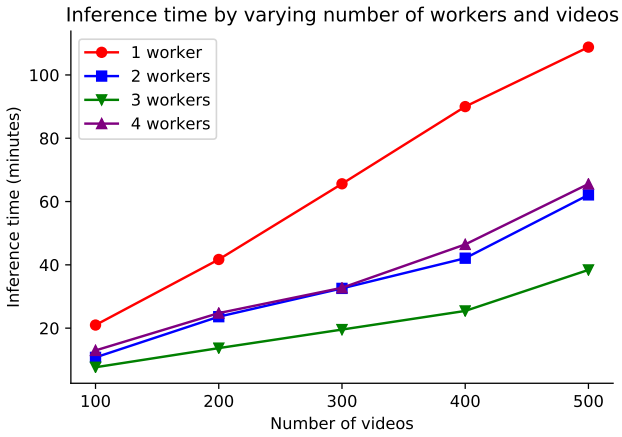

Figure 5. Inference time for a subset of videos with different numbers of workers. The system with 3 parallel workers has the lowest inference time.

4.3.1 *Scalability study*

We ran all the experiments in a Google Colaboratory[10] environment which comes equipped with an NVIDIA Tesla T4 GPU running CUDA version 11.2. We conducted experiments on 500 videos and reported the results with 1, 2, 3, and 4 parallel processes respectively. From Figure 5, we can see that our system is able to scale out as we increase the number of processes. For a single process, the inference time increases linearly. Note that for the single GPU setting with the specified configurations, we observe the best results when the number of parallel workers is 3. We believe this is due to the fact that each separate process creates its own copy of the CLIP model (all on GPU) which leads to high stress on single GPU VRAM.

[10] `https://colab.research.google.com/`

4.3.2 *Qualitative retrieval results*

Figures 3 and 4 show tags and their corresponding frames from videos that have been processed using our system. Notably, as illustrated in Figure 3, given an example video as input, the output of the system is a set of tags ("`European green lizard`", "`lorikeet`", "`ostrich`", "`riding elephant`", "`bathing dog`"), which we have visualized by providing the corresponding frame from the video. These tags are stored in a database along with the video ID, allowing us to retrieve arbitrary videos based on their constituent tags. For instance, you can retrieve the video depicted in Figure 3 by constructing a query that returns all videos in the database that contain say, "`lorikeets`". Closer inspection of the last frame of Figure 3 reveals important nuances of the machine learning model used for tagging. Particularly, "`[bathing dog]`" is an incorrect tag for the last frame, as the frame actually depicts a bathing bear. Mistakes like these can be due to video artifacts such as motion blur, making it hard for the tagging model to crisply and correctly identify frame content. We briefly touch upon potential ways to alleviate this in Section 5.1.

A core component of our video processing system is to be able to retrieve relevant videos based on a query which we demonstrate in Figure 4. We used a conditional "AND" query to retrieve the subset of processed videos that contain two target tags ("`driving car`" and "`car mirror`"). Upon querying the database with this query, several videos with the given tags were returned, of which we randomly selected two videos: Video A (top row) and Video B (bottom row). Importantly, both videos contain pertinent frames corresponding to the target tags of our query, showing the effectiveness of retrieval. Other tags that are not part of the query, but are retrieved by the system include "`[changing gear in car]`" for Video A and "`[news anchoring]`" for Video B. Without the loss of generality, we have illustrated conditional querying for only two predicates, however, it can easily be extended to multiple predicates with different logical conditions.

5 Conclusion

We have presented our approach in building a distributed video processing system that is able to analyze and extract tags from a large corpus of videos. Although this can be done sequentially, by analyzing one video after the other, it is an *embarrassingly parallel* task that can be achieved in a distributed manner. Owing to this, we leveraged PySpark to construct a pipeline that involves initiating multiple parallel jobs, where each job narrowly focuses on the tagging of only a subset of videos, after which the tags are stored in a database for further analysis. Along the way, we present inference time results as well as a qualitative analysis of the retrieval capability of our system.

5.1 Future work

For our experiments, we only used a single GPU instance which places a limit on the number of partitions we can use due to memory limits. Looking into the future, we can extend our work by distributing partitions across multiple devices *and* multiple GPUs. We believe this dimension of horizontal scalability will reconcile any performance inconsistencies depicted in Figure 5.

To improve the ease of use of the system, another direction of work can be to incorporate a language model to interact with the video tag database using natural language. For instance, instead of using a query that requires knowledge of PySpark SQL, a user can simply construct queries such as "Retrieve all videos which contain dogs and cats, but not elephants."

As alluded to in Section 4.3.2, in order to improve tagging precision, one avenue of research can involve using an alternative machine learning model for tagging. Such a model can perhaps take the temporal context of sampled frames into account to improve the quality of tags.

6 Team Contributions

- PyTorch class implementation for inference and tagging: Siddhant Garg and Sridhama Prakhya
- Integrating PyTorch with PySpark to deploy the inference pipeline: Siddhant Garg
- Integrating inference pipeline with MongoDB using PySpark SQL for insertion and querying: Sridhama Prakhya

References

Aote, S. S. and Potnurwar, A. An automatic video annotation framework based on two level keyframe extraction mechanism. *Multimedia Tools and Applications*, 78(11): 14465–14484, 2019.

Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., and Joulin, A. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9650–9660, 2021.

Cha, Y.-J., Chen, J. G., and Büyüköztürk, O. Output-only computer vision based damage detection using phase-based optical flow and unscented kalman filters. *Engineering Structures*, 132:300–313, 2017.

Chang, S.-F., Chen, W., and Sundaram, H. Videoq: a fully automated video retrieval system using motion sketches. In *Proceedings Fourth IEEE Workshop on Applications of Computer Vision. WACV'98 (Cat. No. 98EX201)*, pp. 270–271. IEEE, 1998.

Das, S., Banerjee, M., and Chaudhuri, A. An improved video key-frame extraction algorithm leads to video watermarking. *International Journal of Information Technology*, 10(1):21–34, 2018.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Ilyas, S. and Rehman, H. U. A deep learning based approach for precise video tagging. In *2019 15th International Conference on Emerging Technologies (ICET)*, pp. 1–6. IEEE, 2019.

Kay, W., Carreira, J., Simonyan, K., Zhang, B., Hillier, C., Vijayanarasimhan, S., Viola, F., Green, T., Back, T., Natsev, P., et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017.

Li, B., Weinberger, K. Q., Belongie, S., Koltun, V., and Ranftl, R. Language-driven semantic segmentation. *arXiv preprint arXiv:2201.03546*, 2022.

Liu, X., He, G.-F., Peng, S.-J., Cheung, Y.-m., and Tang, Y. Y. Efficient human motion retrieval via temporal adjacent bag of words and discriminative neighborhood preserving dictionary learning. *IEEE Transactions on Human-Machine Systems*, 47(6):763–776, 2017.

Luo, Y., Zhou, H., Tan, Q., Chen, X., and Yun, M. Key frame extraction of surveillance video based on moving object detection and image similarity. *Pattern Recognition and Image Analysis*, 28(2):225–231, 2018.

Minaei, S., Kiani, S., Ayyari, M., and Ghasemi-Varnamkhasti, M. A portable computer-vision-based expert system for saffron color quality characterization. *Journal of applied research on medicinal and aromatic plants*, 7:124–130, 2017.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pp. 8748–8763. PMLR, 2021.

Saoudi, E. M. and Jai-Andaloussi, S. A distributed content-based video retrieval system for large datasets. *Journal of Big Data*, 8(1):1–26, 2021.

Wang, H. and Schmid, C. Action recognition with improved trajectories. In *Proceedings of the IEEE international conference on computer vision*, pp. 3551–3558, 2013.

Zellers, R., Lu, J., Lu, X., Yu, Y., Zhao, Y., Salehi, M., Kusupati, A., Hessel, J., Farhadi, A., and Choi, Y. Merlot reserve: Neural script knowledge through vision and language and sound. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16375–16387, 2022.

Zhang, Y., Tao, R., and Wang, Y. Motion-state-adaptive video summarization via spatiotemporal analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(6):1340–1352, 2016.

Zhou, Y., Habermann, M., Xu, W., Habibie, I., Theobalt, C., and Xu, F. Monocular real-time hand shape and motion capture using multi-modal data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5346–5355, 2020.